DOCUMENT RESUME

ED 317 049 FL 018 387

AUTHOR Morgan, Jerry L.
TITLE A Tool for Displaying Syntactic Trees.
INSTITUTION Illinois Univ., Urbana. Language Learning Lab.
REPORT NO LLL-T-9-86
PUB DATE Jul 86
NOTE 22p.
AVAILABLE FROM Language Learning Laboratory, University of Illinois at Urbana-Champaign, G70, Foreign Languages Building, 707 South Mathews, Urbana, Illinois 61801.
PUB TYPE Reports - Evaluative/Feasibility (142)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computer Software; *Programing; *Structural Analysis (Linguistics); *Syntax
IDENTIFIERS *Parsing; Texas Instruments

ABSTRACT

A computer program for drawing syntactic phrase markers as trees is described. The program was developed for use on Texas Instruments Explorer Lisp machines. The tree is drawn by recursive descent, left to right. The tree-drawing function takes two arguments: (1) an atom constituting the tree, and (2) a font specification to be used in drawing the tree. The tree argument must be an atom that constitutes the root node of the tree to be displayed. For interfacing to parsers or other tools that represent parses as lists, a function can be used to convert a list representation of a parse to the linked-atom representation used in this program. Two convenient elaborations are in development: a mouse handler for the tree window and a feature display. Appended materials include some sample trees from the program and the program's source listing. (MSE)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED317049

Language Learning Laboratory

LLL

LLL-T-9-86
JULY 1986

The School of Humanities

University of Illinois
at Urbana-Champaign

A TOOL FOR DISPLAYING SYNTACTIC TREES

BY JERRY L. MORGAN

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
C.C. Cheng
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☑ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

FL018387

BEST COPY AVAILABLE

LANGUAGE LEARNING LABORATORY
School of Humanities
University of Illinois at Urbana-Champaign

Technical Report No. LLL-T-9-86

A TOOL FOR DISPLAYING SYNTACTIC TREES

Jerry L. Morgan

Associate Professor of Linguistics
University of Illinois, Urbana-Champaign

July 1986

Available from: Language Learning Laboratory, University of Illinois at Urbana-Champaign, G70 Foreign Languages Building, 707 South Mathews, Urbana, Illinois 61801.

Introduction

This report is a description of a program for drawing syntactic phrase markers as trees, developed for use on Texas Instruments Explorer Lisp machines. The Common Lisp source is included as an appendix, along with some illustrations of the output of the tree-drawing program.

The source code may be obtained either as an ASCII file on an IBM PC formatted 5 1/4" floppy disk, or as an Explorer file on a 1/4" Explorer tape cartridge. Requests, together with tape or formatted disk, should be sent to:

J. L. Morgan
Department of Linguistics
University of Illinois
4088 Foreign Languages Building
707 S. Mathews
Urbana, IL 61801

The work described in this report was made possible by a gift from the Texas Instruments Corporation to the University of Illinois.

Background

The need for this program arose during efforts to implement a natural language parser on Lisp machines. In the process of this work, it soon became obvious that a tree-drawing facility was indispensable. In most current theories, syntactic structure is of a form that can be represented by labelled bracketings, or, entirely equivalently, by 'trees'. The equivalence notwithstanding, labelled bracketings are not effective visual representations for anything beyond the most trivial kind of structure. So the ability to represent syntactic structures graphically as trees is crucial, not only for a parser but for any computational tool intended for use by syntacticians.

To be most useful, a tree-drawing program should display the tree in a maximally readable way. In this program two decisions were made to increase readability, based on the author's judgments of which alternatives were most readable: first, all sister non-terminal nodes are printed on the same level. Second, all terminal nodes--that is, the words in the sentence--are printed at the same level.

The Algorithm and Data

The tree-drawing algorithm used here is quite simple: the tree is drawn by recursive descent, left to right, as follows:

Calculate the constant amount of fill space (= FILL) between terminal elements from two factors: the print length of the combined terminal elements of the entire tree, and the width of the display area. Initialize the terminal cursor (= CURSOR) to FILL.

For each node, starting at the left:

> If the node is TERMINAL, print it at the bottom of the display, at the position of CURSOR. Update CURSOR by setting to FILL spaces beyond the right end of the terminal element just printed.
>
> If the node is NOT TERMINAL, first print its daughters. Then print this node, with its position determined as follows:

Its height in the display is proportional to its depth in the tree.

Its horizontal position in the display is determined as follows:

If it has an odd number of daughters, print it centered over its middle daughter.

If it has an even number of daughters, print it centered over a point mid-way between the left edge of its first daughter and the right edge of its last daughter.

Finally, if the node has daughters, draw a line from the node's bottom midpoint to the top midpoint of each daughter.

The tree-drawing function (draw-tree) takes two arguments: first (required) an atom constituting the tree to be drawn, and second (optional) a font specification to be used in drawing the tree. If the font argument is missing, the program chooses a font based on the size of the tree to be displayed.

The tree argument must be an atom that constitutes the root node of the tree to be displayed. It must have the following data in its property list, as must all nodes in the tree:

Property Name	Value	Purpose
mother	atom	pointer to immediately dominating node (or nil, if it is the root node)
daughters	list of atoms	pointer to daughter nodes (OR nil, if it is a terminal node)
level	integer	height of this node in the tree, starting with 1 at the root and increasing with the depth of the node (all terminal nodes have the same level value: the maximum depth of the tree)
category	atom	if terminal, then the word; else a syntactic category
length	integer	if terminal, string length of the word; if non-terminal, the sum of string lengths of the node's terminal descendants
nr-of-terminals	integer	if terminal, 1; else the number of terminal descendants of this node

For interfacing to parsers or other tools that represent parses as lists, the function *maketree* can be used to convert a list representation of a parse to the linked-atom representation used by *draw-tree*. *Maketree* takes a single argument: a list representation of a parse, in the fashion of a labelled bracketing. The *car* of each sub-list must be an atom representing the syntactic category; the *cdr* must be a sequence of one or more elements. Each of these elements may be either an atom (if the element is terminal) or a sub-list (if non-terminal). For example, figure one shows a list expression and figure two shows the corresponding tree.

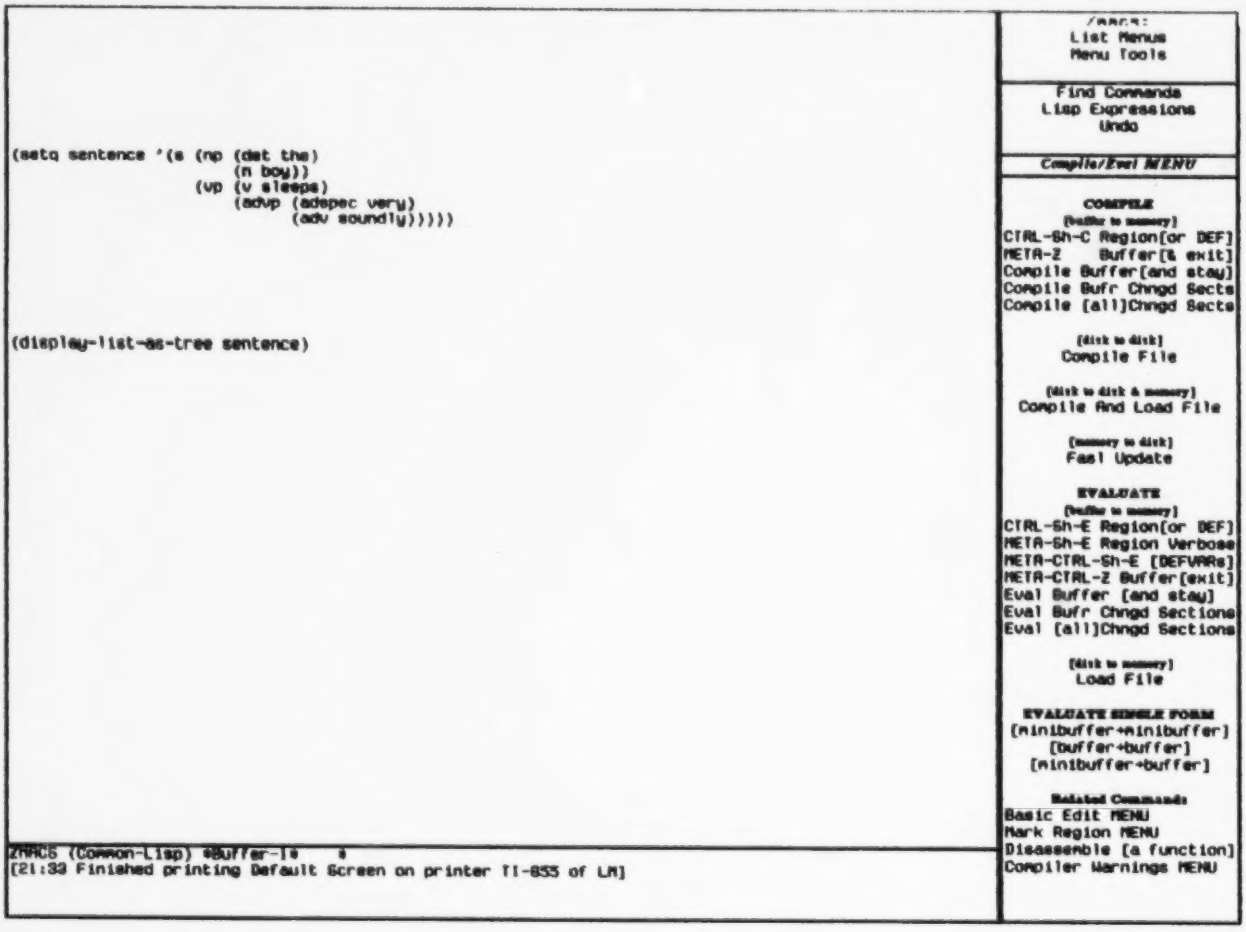


FIGURE ONE

BEST COPY AVAILABLE

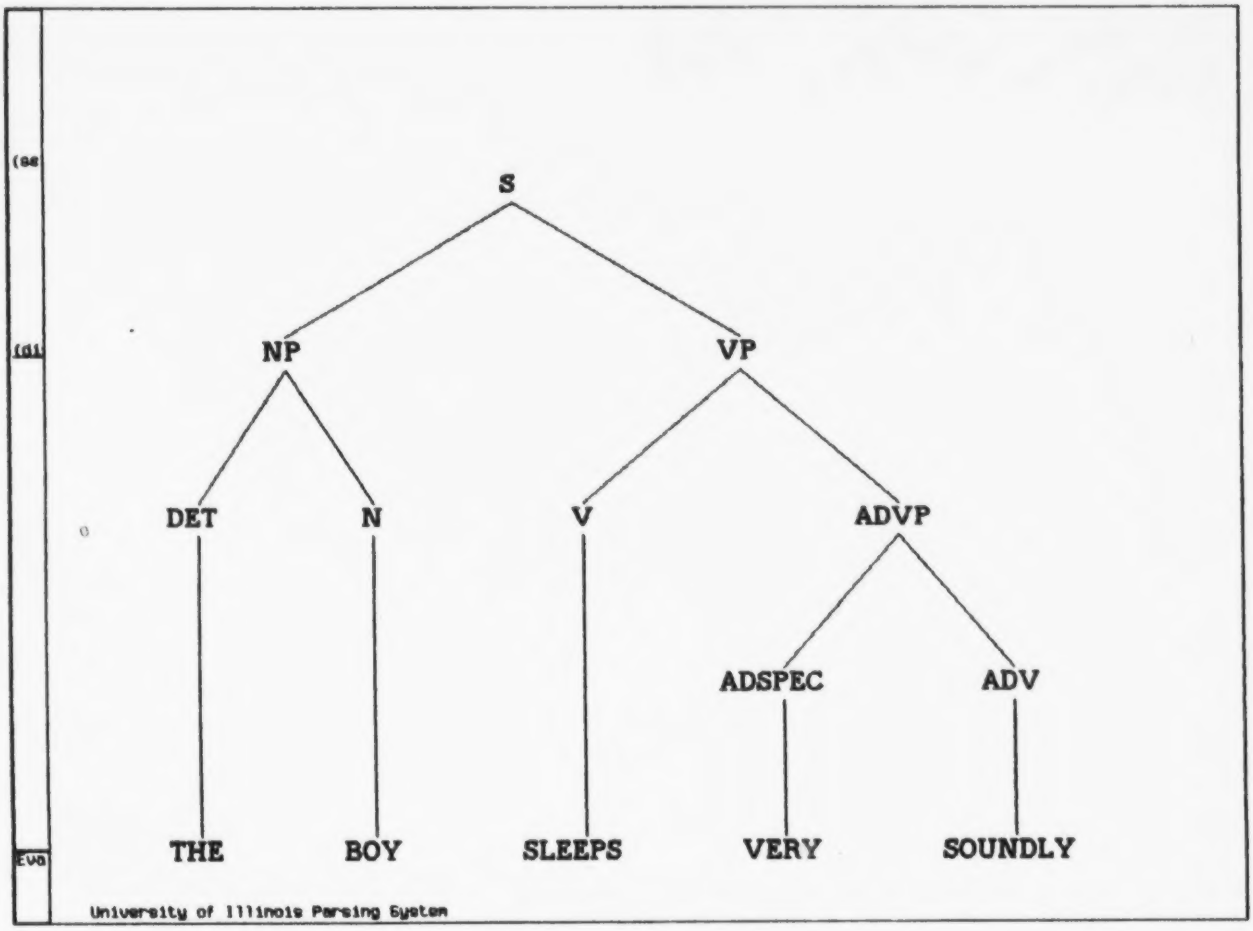

S
NP
VP
DET
N
V
ADVP
ADSPEC
ADV
THE
BOY
SLEEPS
VERY
SOUNDLY
University of Illinois Parsing System

FIGURE TWO

Using the Tree-Display

The tree-drawing functions can be called with either of the two kinds of representations discussed above as arguments. If the parse is in the form of a list, then the tree can be drawn by entering

> (display-list-as-tree parse)

This allows the tree-drawing program to choose the font. Or, entering

> (display-list-as-tree parse font)

(where font is a font specification) specifies the font to be used. The function *display-list-as-tree* first converts the list to a set of atoms, then passes the root node atom to the function *draw-tree*.

If the parse is already encoded in a set of atoms and properties as discussed above, then *draw-tree* may be called directly, with the root node as first argument and a font as the (optional) second argument.

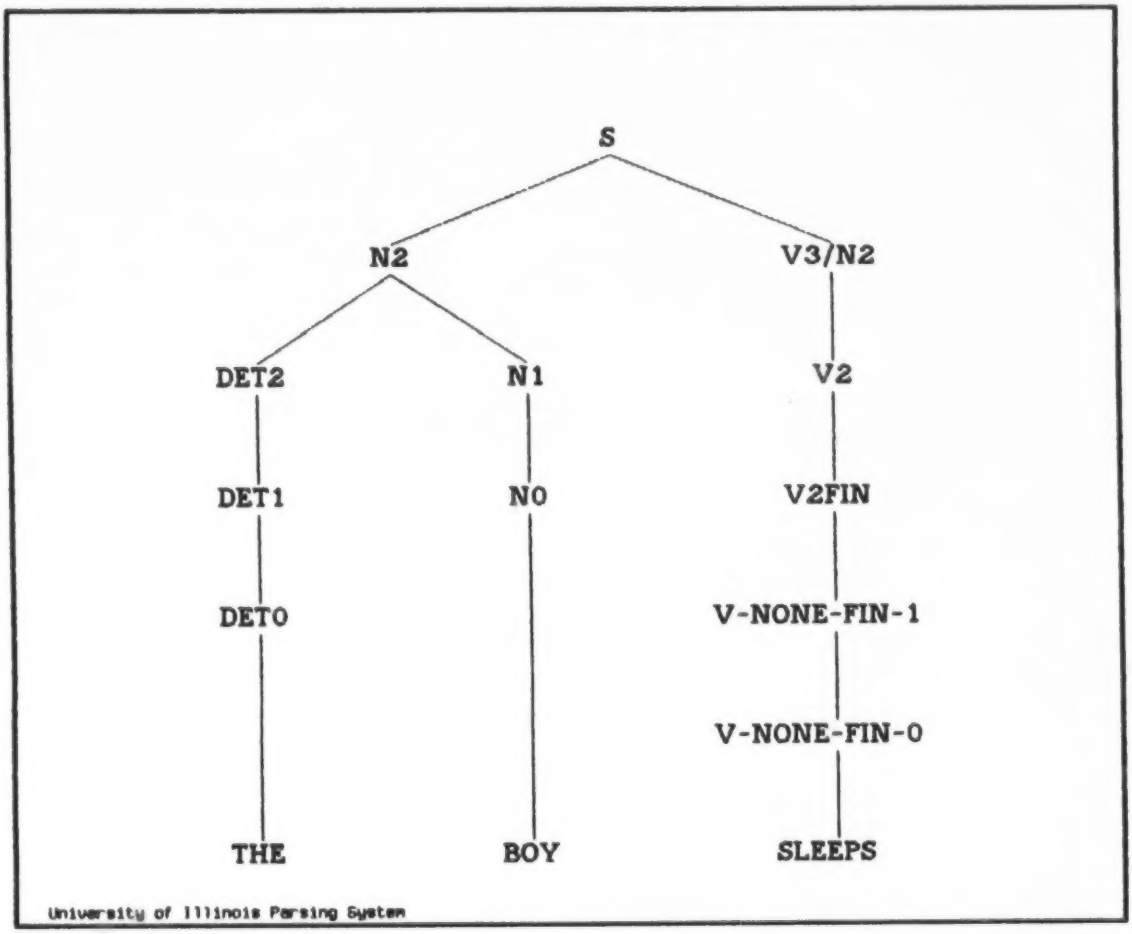
The program will then open a window covering most of the screen, leaving a small margin uncovered on the left. The tree will be drawn on this window, which will remain on the screen until the user clicks the mouse outside the tree window (in the uncovered space on the left). Then the tree window disappears and the previous window is uncovered.

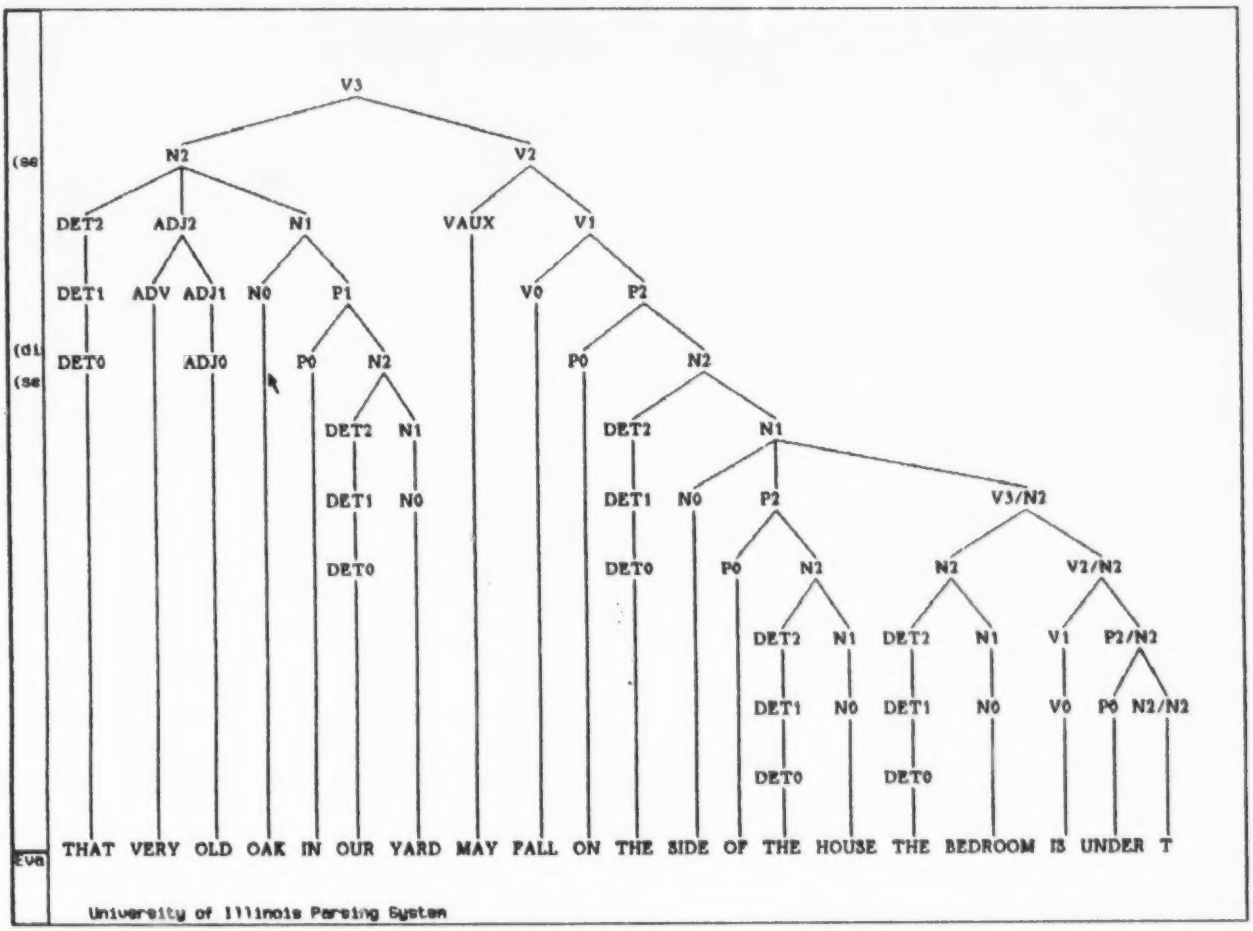
Future Developments

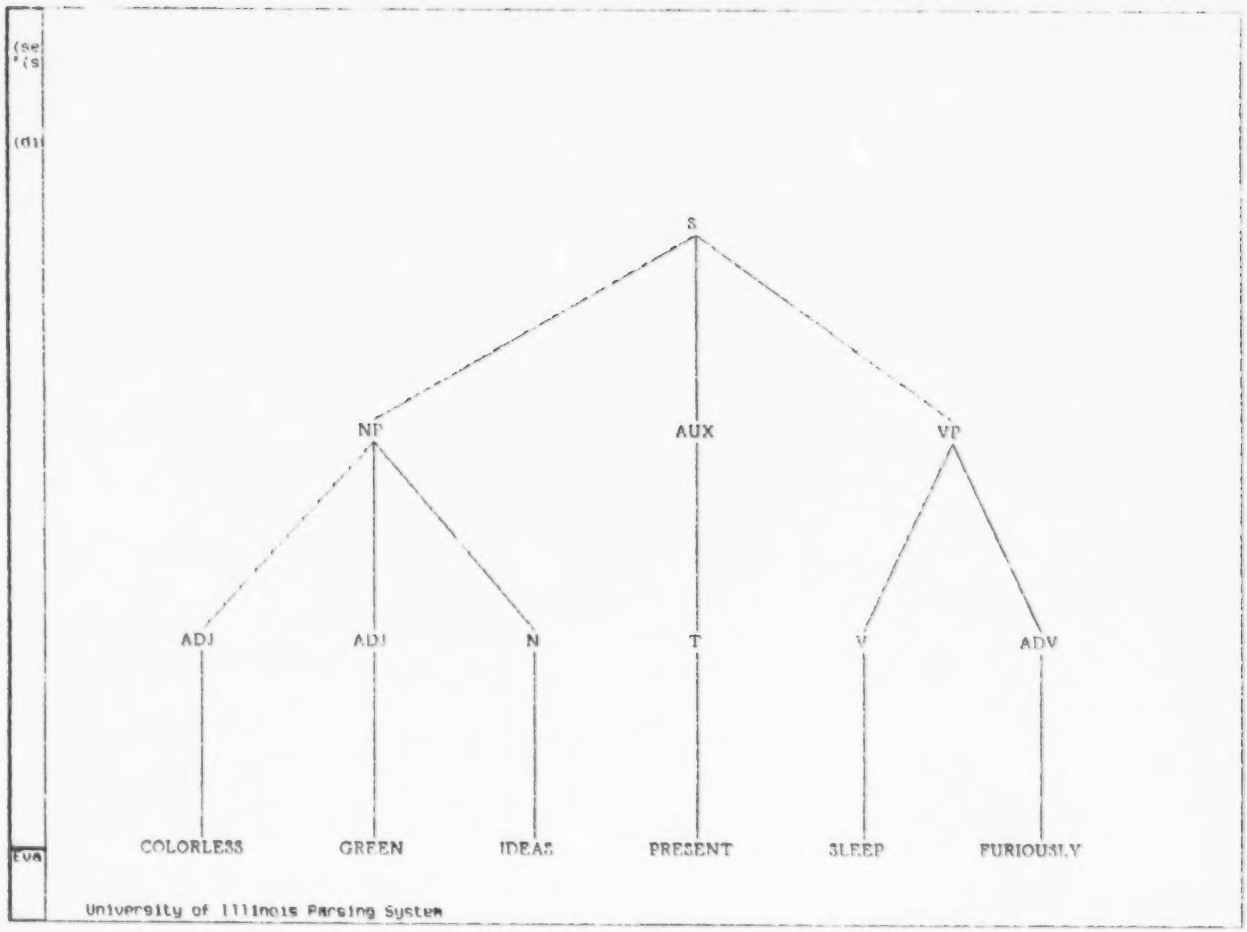
Two convenient elaborations are under development: a mouse handler for the tree window, and a feature display. It would be useful to allow the user to choose a node in the display with the mouse, and examine undisplayed properties of the chosen node; in particular, to display syntactic features of a chosen node in temporary windows superimposed on the tree window.

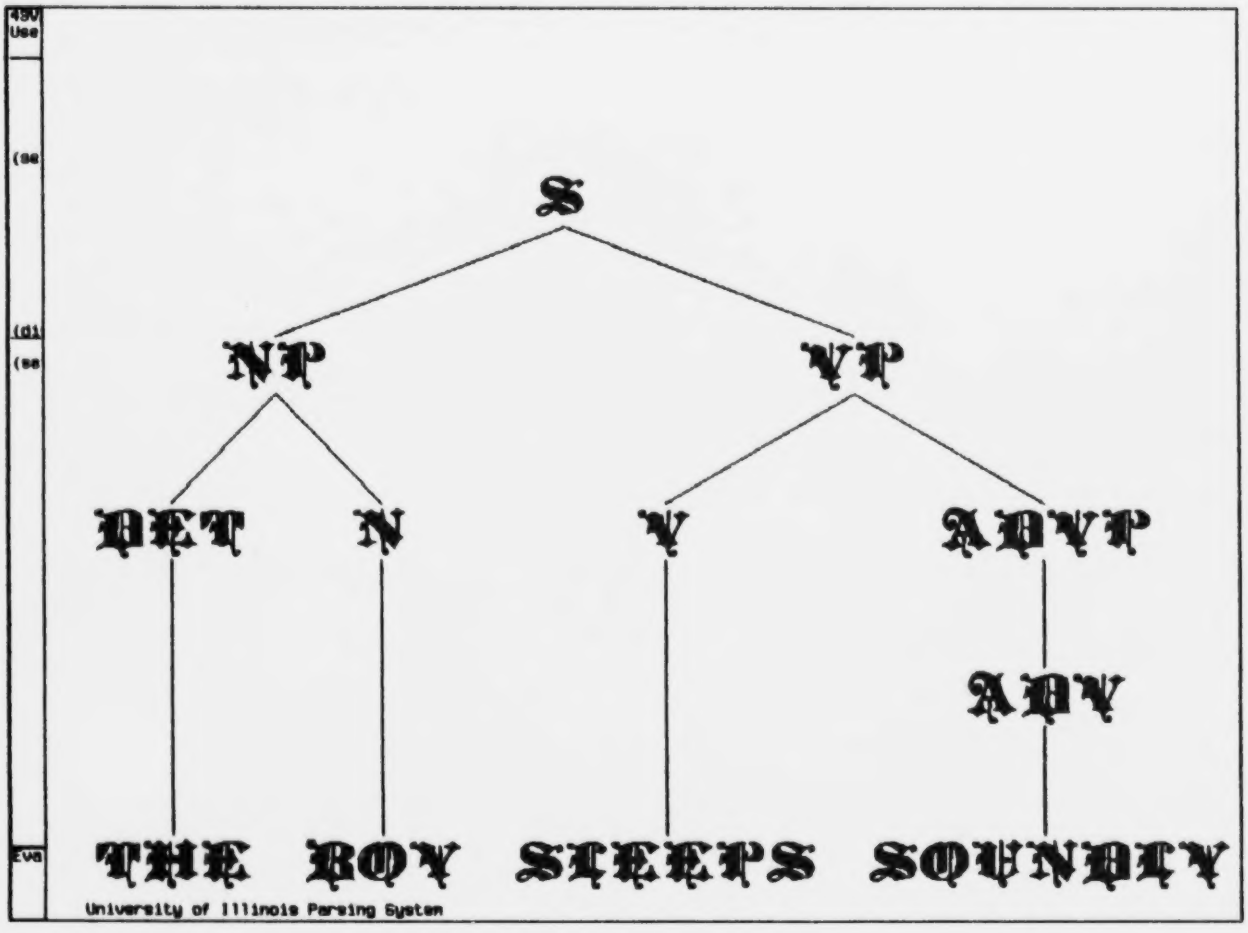
APPENDIX A

Some Example Trees from the Program


S
N2
V3/N2
DET2
N1
V2
DET1
N0
V2FIN
DET0
V-NONE-FIN-1
V-NONE-FIN-0
THE
BOY
SLEEPS
University of Illinois Parsing System


V3
N2
V2
DET2
ADJ2
N1
VAUX
V1
DET1
ADV
ADJ1
N0
P1
V0
P2
DET0
ADJ0
P0
N2
P0
N2
DET2
N1
DET2
N1
DET1
N0
DET1
N0
P2
V3/N2
DET0
DET0
P0
N2
N2
V2/N2
DET2
N1
DET2
N1
V1
P2/N2
DET1
N0
DET1
N0
V0
P0
N2/N2
DET0
DET0
THAT VERY OLD OAK IN OUR YARD MAY FALL ON THE SIDE OF THE HOUSE THE BEDROOM IS UNDER T
University of Illinois Parsing System


(se
'(s
(d1
S
NP
AUX
VP
ADJ
ADJ
N
T
V
ADV
COLORLESS
GREEN
IDEAS
PRESENT
SLEEP
FURIOUSLY
eva
University of Illinois Parsing System


S
NP
VP
DET
N
V
ADVP
ADV
THE
BOY
SLEEPS
SOUNDLY
University of Illinois Parsing System

APPENDIX B

Source Listing of the Program

```
;;;  -*-  Mode: Common-Lisp;  Base: 10  -*-
;;;  This file contains functions for drawing Phrase-marker "trees" on a Texas Instruments Explorer.
```

Copyright (C) 1986, Department of Linguistics, University of Illinois. All rights reserved.

```
;;;
;;;     The two main functions in this file are "maketree" and "draw-tree".  "Maketree" takes a parse
;;;     in list form, in the fashion of a labelled bracketing.  It converts it to a data structure used by
;;;     "draw-tree".  "Draw-tree" draws a phrase-marker display in a graphics window.  This version
;;;     does not require the Explorer graphics toolkit.
;;;
```

```
;;;
;;;     "display-list-as-tree" is the interface to the parser.
;;;        The parser calls it with two arguments:
;;;           1)  a parse in the form of a list (see examples at the end of this file)
;;;           2)  a font.  This argument is optional; if not supplied, "draw-tree" will choose a font
;;;               depending on the length of the sentence to be parsed.
;;;
```

```
(DEFUN display-list-as-tree (tree &optional font-spec)
  (IF (VARIABLE-BOUNDP tree-window)
      (SEND tree-window :expose)
      (open-tree-window))
  (draw-tree (maketree tree)
             font-spec))
```

```
;;;
;;;     Use gentemp-list to keep track of gentemps so they can be gotten rid of later
;;;
```

```
(setq gentemp-list nil)
```

```
;;;
```

```
        (PUTPROP result                                  ; combined length of terminals
           (APPLY #'+
             (MAPCAR #'(LAMBDA (z) (GET z 'length))
                (GET result 'daughters)))
           'length)
        (PUTPROP result                                  ; number of terminals
           (APPLY #'+
             (MAPCAR #'(lambda (z)
                    (GET z 'nr-of-terminals))
                (GET result 'daughters)))
           'nr-of-terminals)))
  result))                                               ; return the gentemp'ed
                                                         ; atom that is the root node
```

```
;;;
```

```
(DEFFLAVOR tv:tree-window ()
        (
         tv:select-mixin
         tv:graphics-mixin
         tv:stream-mixin
        tv:borders-mixin
         tv:label-mixin
         tv:minimum-window
         ))
```

```
;;;
```

```
(LET* ((max-depth (max-depth-of-tree root))
       (length (GET root 'length))                    ; get height of tree
       (font                                          ; font used in printing
         (COND (font-spec                             ; syntactic category
                (font-spec)
                ((<length 30)
                 fonts:mets
                ((<length 105)
                 fonts:tr12
                ((> length 200)
                 fonts:tr10)
                (t fonts:tr8)))
       (char-height (- (tv:font-char-height font) 5))
       (height (QUOTIENT bottom                       ; vertical space alloted for
                         (* 1.1 max-depth)))          ; each line of text
       (width (SEND tree-window                       ; typical character width, used
                    :character-width                  ; in calculating placement
                    #x58
                    font ))
       (fill (MAX 4
                  (TRUNCATE (- right                  ; fill is the amount of space
                               left                   ; inserted between words
                            (* width
                               (GET root 'LENGTH)))
                         (1+ (GET root  'nr-of-terminals)))))
       (offset (MAX 9 fill)))                         ; offset is the cursor
      (DECLARE (SPECIAL font)                         ; position for printing
               (SPECIAL char-height)                  ; strings
               (SPECIAL width)
               (SPECIAL height)
               (SPECIAL fill)
               (SPECIAL offset))
      (draw-2 root))))                                ; draw the new tree
```

;;;

```
(IF daughters
        (draw-node-lines root))))                    ; Draw mother-daughter lines
```

```
;;;
```

```
            (TRUNCATE (* height level))                    ; height of node
             nil                                           ; in the window
             nil
             font
             tv:alu-ior)
          ;;; else (it has no daughters)
          ;;; print it beginning at the current
          ;;; cursor (offset) position.
            (SEND tree-window
                :string-out-explicit
                name
                (SETQ x-start offset)                      ; save for later
                (TRUNCATE (* height level))
                 nil
                 nil
                 font
                 tv:alu-ior))
  ;;; Record the following information on this node's property list:
  (PUTPROP node x-start 'x-start)                          ; left end on the screen
  (PUTPROP node x-end 'x-end)                              ; right end
  (PUTPROP node (+ 4 y-end char-height) 'bottom)           ; bottom
  (PUTPROP node
           (+ x-start
              (TRUNCATE length
                        2))
           'mid)
  (PUTPROP node (- y-end 5) 'top)                          ; top
  (IF (NULL daughters)                                     ; if this is a terminal node, update
      (SETQ offset (+ x-end fill))))))                     ; the cursor
```

;;;

```
;;;
;;;     "Draw-node-lines" draws lines in the window between mother and daughters.
;;;     The line originates at mid-bottom of mother.
;;;     IF there's more than one daughter, THEN it draws from middle of mother to middle of each
;;;     daughter.  ELSE it uses mother's mid-point for mother AND daughter.
;;;
```

```
(DEFUN draw-node-lines (root)
   (LOOP for daughter in daughters do
         (SEND tree-window
               :draw-line
               (get root 'mid)
               (get root 'bottom)
               (IF   (CDR daughters)
                     (get daughter 'mid)
                     (get root 'mid))                    ; to avoid crooked verticals.
               (get daughter 'top)
               tv:alu-ior)))                             ; width of line = 1 pixel
```

```
;;;
;;;     A sample tree and function call, for demo and testing
;;;
;;;     (setq s '(s
;;;           (np  (det the)
;;;                (n1  (n boy)
;;;                     (rel  (comp that)
;;;                          (s/np  (np  (det the)  (n1  (n girl)))
;;;                                 (vp/np  (v loves))))))
;;;           (vp  (v told)
;;;                (np  (det my)(n1  (n friend)))
;;;                 sbar  (comp that)
;;;                      (s  (np  (det your)(n1  (n dog)))
;;;                          (vp  (v eats)
;;;                               (np  (n1 cats))
;;;                               (pp  (p with)
;;;                                    (np  (det a)
;;;                                         (n1  (n spoon)))))))))
;;;
;;;     (draw-tree (maketree s))
;;;
```